High-Level Design of Amanzi, the Multi-Process High Performance Computing Simulator

January 11, 2012

ASCEM-HPC-2011-03-1

ASCEM

United States Department of Energy

Environmental Management

safety ❖ performance ❖ cleanup ❖ closure

D. Moulton, LANL
M. Berndt, LANL
M. Buksas, LANL
R. Garimella , LANL
L. Pritchett-Sheats, LANL
G. Hammond, PNNL
M. Day, LBNL
J. Meza, LBNL

DISCLAIMER

Printed in the United States of America

Prepared for
U.S. Department of Energy

Authorization:

______________________________	01/11/2012
Dr. Paul Dixon, Los Alamos National Laboratory Multi-Lab ASCEM Program Manager	Date
______________________________	01/11/2012
Dr. David L. Brown, Lawrence Berkeley National Laboratory ASCEM Technical Integration Manager (Acting)	Date
______________________________	01/11/2012
Dr. J. David Moulton, Los Alamos National Laboratory ASCEM Multi-Process HPC Simulator Thrust Lead	Date
______________________________	01/11/2012
Dr. Carl I. Steefel, Lawrence Berkeley National Laboratory ASCEM Multi-Process HPC Simulator Deputy Thrust Lead	Date

Concurrence:

______________________________	______________
Dr. Mark Williamson, ASCEM Program Manager EM-32 Groundwater & Soil Remediation	Date
______________________________	______________
Kurt Gerdes, EM-32 Office Director for Groundwater & Soil Remediation	Date

Table of Contents

Executive Summary

The Advanced Simulation Capability for Environmental Management (ASCEM) program is developing an approach, and the supporting open-source tools, to understand the fate and transport of contaminants in natural and engineered systems. The multi-process High Performance Computing (HPC) simulator, named Amanzi, provides a flexible and extensible simulation capability for ASCEM. The goal of this document is to establish a common understanding of the high-level design strategy for all stakeholders in ASCEM as well as to describe the overall design strategy. However, explicit implementation details are beyond the scope of this document, and will be addressed in future Amanzi documentation. The intended audience for this document includes *Amanzi* developers, members of the Platform and Site Applications Thrusts, and other parties interested in the ASCEM project.

This design document begins with a brief overview of the goals of the ASCEM project, and its organizational structure. In particular, this discussion establishes the need for a flexible and extensible open-source simulation capability. Next a high-level description of the hierarchical object-oriented design of Amanzi is presented and key terminology is introduced. Specifically, in ASCEM a *process model* refers to a complete mathematical description of a physical or bio-geochemical process. A specific subsurface flow and reactive transport scenario of interest to DOE-EM is then described by a set of coupled process models. To capture this modular view of coupled processes, the design introduces the concept of a Multi-Process Coordinator (MPC), and defines a Process Kernel (PK) as the discrete representation of these process models. The lower-level toolsets that support the PKs are also discussed, namely, the mesh infrastructure, discretization, reaction, and solvers toolsets are highlighted. To effectively model sites with complex hydrostratigraphy, as well as engineered systems, a dual unstructured/structured mesh capability is being developed. The high-level concepts needed to support these two approaches are presented. Significant capabilities are available in existing Third Party Libraries (TPLs) and are leveraged in Amanzi. The role of key TPLs, such as Trilinos and BoxLib, is discussed. Finally, a high-level overview of the Platform Toolset, Akuna, is presented, and its role in driving Amanzi is discussed.

1 Introduction

This document describes the high-level design strategy for *Amanzi*, the Multi-Process High Performance Computing (HPC) Simulator for the Advanced Simulation Capability for Environmental Management (ASCEM) program. The goal of ASCEM is to develop a transformational approach and state-of-the-art scientific tools for integrating data, software, and scientific understanding to predict the fate of contaminant transport in the subsurface. The program combines open-source high performance computing algorithms, models, data analysis and integration approaches, and an evolving understanding of subsurface hydrological-biogeochemical processes. ASCEM will also provide other DOE programs, as well as the overall scientific community, with a code that will be applicable to a variety of subsurface flow and transport scenarios. It is envisioned that the community code will be updated and augmented as new scientific insights are developed through DOEs research programs in the Office of Advanced Scientific Computing Research (ASCR) and the Office of Biological and Environmental Research (BER), as well as other federal research programs.

The ASCEM program is organized into three Thrust Areas: 1) High Performance Computing (HPC) Multi-Process Simulator, 2) Platform and Integrated Toolsets, and 3) Site Applications. The HPC Thrust includes structured and unstructured meshing approaches; new solvers for coupled physical and geo-biochemical processes; advanced methods of discretization in time and space; and capabilities to select, and coordinate the use of problem-specific processes. The Platform Thrust provides toolsets that facilitate model setup and analysis, parameter estimation and uncertainty quantification, risk assessment and decision support, information and data management, and visualization in a consistent and flexible user interface and modeling work flow. The Site Application Thrust coordinates and implements demonstrations through working groups to provide data and feedback to developers and to ensure that the software is developed in a manner that will engage users and benefit DOE EMs remediation obligations.

Amanzi is the flexible and extensible computational engine for ASCEM that will simulate the coupled processes described by the conceptual models developed using the ASCEM Platform, *Akuna*. These conceptual models span a range of process complexity in flow and reactive-transport. Detailed mathematical descriptions of these models are provided in [16].

The intended audience for this document includes *Amanzi* developers, members of the Platform and Site Applications Thrusts, and other parties interested in the ASCEM project. The goal of this document is to establish a common understanding of the high-level design strategy for all stakeholders in ASCEM as well as to describe the overall design strategy. This document does not, however, contain explicit implementation details. Priorities, such as the relative importance of particular process models, or details of the interface between *Amanzi* and *Akuna*, are not addressed in this document, except for a brief summary.

A critical aspect of the ASCEM project is to provide tools that support a graded and iterative approach to conceptual model development. To meet this goal, the *Amanzi* design incorporates the following high-level goals:

- Provide a set of robust and numerically convergent discrete representations of the coupled processes that describe processes, including flow and reactive transport, in porous media.
- Provide a dual set of capabilities that allow for both a structured and an unstructured mesh option.

- Accelerate the development and verification of a complete end-to-end simulation capability by leveraging pre-existing software and numerical algorithms.
- Design for execution on a diverse range of modern computing hardware, taking optimal advantage of high-performance features.
- Provide a well-defined interface between *Amanzi*, the HPC Simulator, and *Akuna*, the ASCEM Platform toolset.

One of the main strategies for achieving these goals is to use a modular object-oriented design. Modularity will allow ASCEM software developers to quickly prototype and develop new capabilities as needed by site users. In addition, by clearly defining application programming interfaces, ASCEM will allow the inclusion of new processes and algorithms from researchers in the community thereby creating an extensible community code. The use of an object-oriented programming (OOP) model will also allow for a more flexible simulator. An excellent review of modern development and object-oriented design methods is given in [11]. The toolsets (described in the following sections) for both the physical processes and the numerical methods will develop APIs (Application Programming Interfaces) to define how they provide access to their data and methods. With this approach, control and responsibility are clearly defined, and new models or algorithms can be adopted without refactoring large sections of the code. Although the *Amanzi* design will use an object-oriented style, it does not require that the lower level functions specifically use an OOP language. This additional freedom will maximize compatibility with other software packages.

In this document, the methodologies undertaken by the ASCEM team to address the above design criteria are presented. First, the basic structure of the HPC simulator is outlined. This discussion includes a description of the interface to the mathematical representation of the problem-specific components of the porous media application. Then an overview of the key algorithmic components of the simulator is given. A discussion of the role of third-party libraries that are included with the *Amanzi* build follows in section 4. Finally, the interface between *Amanzi* and the Platform Toolset, *Akuna* are outlined.

2 High-Level Design of *Amanzi*

Amanzi takes as input a conceptual model, which describes a set of coupled processes such as flow and reactive transport, and evolves these processes in time, generating output for analysis and visualization. To support the graded and iterative approach to conceptual model development that is required by EM, Amanzi must be both modular and extensible. This objective is achieved with a hierarchical and modular design that captures all the steps involved in translating a conceptual model to output for analysis. To support this hierarchical view an object oriented programming model is used, with higher level objects and much of the code being developed in the C++ language. Specifically, the design has high-level objects that abstract process models and their coupling, supporting toolsets that provide the building blocks for these high-level objects, and low-level objects and services that are used by the supporting toolsets. These three component levels are shown schematically in Figure 1.

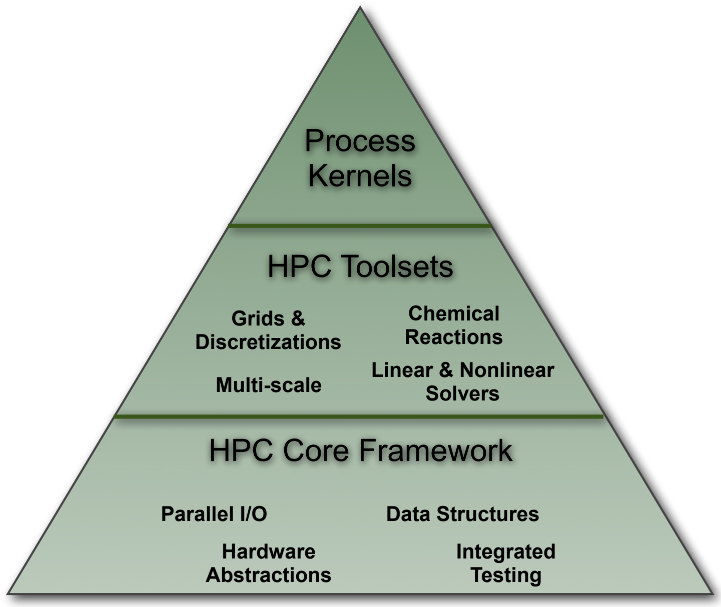

Figure 1: Schematic of the main components of *Amanzi*.

The *Amanzi* simulator will provide the following algorithmic components:

- **Simulation Driver**: Initialize the Multi-Process Coordinator, direct I/O, and instruct the MPC to execute a series of time steps.
- **Multi-process Coordinator (MPC)**: Initialize/maintain state data, implement sequence of process kernels for time step, optionally control solution-adaptive mesh generation, and support recursive implementation.
- **Process Kernels (PK)**: Numerical implementation of physical process models
- **HPC Toolsets** for the implementation of process kernels:
  - Mesh infrastructure: Define/allocate data containers, provide interface to underlying data and communication primitives supporting distributed memory architectures.
  - Discretizations: Low-level toolsets for temporal and spatial derivative approximations
  - Reactions: classes depicting chemical species and reactions on a cell.
  - Solvers: Numerical solution of linear and nonlinear equation systems.
- **HPC Core Framework**: Parallel I/O, data structures, error handling.

The requirements of each of these components is discussed below. An overall schematic of the design is displayed in Figure 2. Implementation of the corresponding software is tied closely with the assumptions imposed by the underlying data structures, and these in turn depend directly on each of the supported mesh options.

2.1 Simulation Driver

The Simulation Driver is responsible for the setup and execution of simulation through the instantiation of the MPC, or the base MPC if there is an MPC hierarchy (discussed below). The Simulation Driver interacts with the input from the Platform to collect the problem setup, computing environment (e.g., processor map), method specification, and create the necessary internal objects. It then instructs the MPC to perform a set of time-steps according to the input, and to dump visualization data at a prescribed sequence of steps. In addition, it creates and passes an observation object, which includes a specification of any requested observation data that the MPC needs to collect, and a container for the MPC to store this data. These observations are small amounts of data needed by various Platform Toolsets, such as Parameter Estimation, to perform their analysis and track simulation progress. For example, the observation data may include a time series of concentrations at a specific location.

2.2 Decomposition of Modeling and Simulation

2.2.1 Multi-process Coordinator

The Multi-Process Coordinator (MPC) marshals the complete numerical approximation to the mathematical problem being solved in order to implement a single discrete time step advance. The MPC time step consists of a sequence of PK calls that compose the complete model; it manages the current state of the system and any additional data associated with the conceptual model (including a copy of the previous state, if necessary), and gathers/dumps this data at the request of the Simulation Driver for visualization and restarts. The MPC additionally gathers the observation data requested from the simulation driver and provides various bookkeeping and recovery functions, such as queries to establish discrete time step sizes compatible with the entire set of PKs, and the management of failed time step attempts (e.g., decreasing the step size and managing a second attempt).

Under the structured grid option, the MPC implements an adaptive mesh refinement (AMR) strategy. A corresponding adaptive capability may eventually be supported for the unstructured grids. For structured grids, AMR is implemented using a recursive mesh generation and sub-cycling integration strategy. When the Simulation Driver constructs the base MPC object, it communicates sufficient information to allow the instantiation of a base mesh object. If automatic meshing is supported, solution errors are estimated over the state and a refined mesh is generated over a subregion of the original domain. A child MPC object is then constructed and maintained by the parent MPC object, and the process repeats until the solution is adequately represented. When the simulation driver requests a time step, the base MPC first coordinates the PKs to effect a time step of the algorithm on the base mesh, but then follows with a recursive call to the child MPC to advance the child mesh over the same time interval. When the child MPC has completed the request, the state data on the composite mesh system is synchronized using special PKs that are managed by the parent MPC. Error estimation procedures are performed on the parent mesh data in order to determine a new set of meshing instructions for the child MPC, and a new child is created by reusing data on overlap with the old child MPC, and otherwise transferring (interpolating) data from the parent mesh. Note that recursion is not mandatory for dynamic mesh refinement in general, but it does

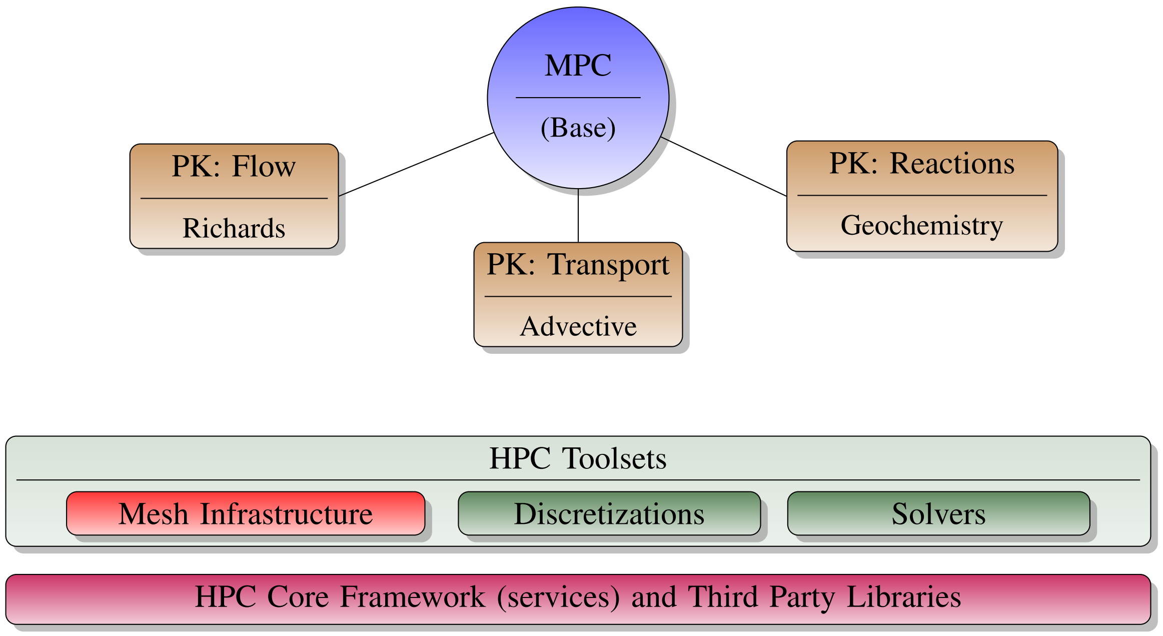

Figure 2: Schematic showing a base Multi-Process Coordinator (MPC) that has instantiated process kernels (PKs) for flow, transport, and reaction. These high-level objects leverage the underlying toolsets, core services, and third party libraries to evolve the numerical simulation in time.

represent a key aspect of the structured grid implementation framework.

2.2.2 Process Kernels

Process Models are the mathematical description of a specific, reasonably independent physical or geo-biochemical phenomena. Each process model will necessarily involve some quantities that it shares with other processes. For example, the concentration of a primary species is involved in both transport and reaction. Process Kernels (PKs) are the software objects that implement a particular numerical approximation to a Process Model, such as flow and transport. Each PK is designed to be self-contained, using only information provided through it's interface to compute an update to the state of the system. Implementation of a PK depends on the mesh in general, and will involve the discretization and solver toolsets to interact directly with the data, as required.

2.3 HPC Toolsets

2.3.1 Mesh Infrastructure

The mesh provides a fundamental data structure that bridges the conceptual site model with the numerical methods, and is ultimately the building block that connects the resulting simulation with the computing hardware. Mesh infrastructure refers to the framework used to store a description of the mesh (physical location and state quantities), implement memory allocation and parallel communication activities that can be tailored specifically to the data layout. The mesh infrastructure provides an API that provides efficient access to mesh entities, and access to underlying state data, as appropriate to the specific meshing model, in order to support the construction of PKs.

Unstructured meshes are an effective approach for capturing the complex geometry of the hydrostratigraphy and topography found at many sites. *Amanzi* will support more general cell types, such as polyhedra, as this is particularly advantageous at pinchouts or fault intersections in three dimensions. On these unstructured meshes, *Amanzi* makes no *a priori* assumptions about the layout of the data in memory maps to physical space. The mesh infrastructure reads a mesh from a file, in serial or parallel; creates the necessary data structures for important mesh entities, such as faces and cells; and provides access to this data through its API. To facilitate parallel simulations on distributed memory systems, the mesh infrastructure manages all connectivity and ghost cell data across processors. At this time, the unstructured option for *Amanzi* supports four different mesh frameworks: STK mesh, which is part of the Trilinos project, MOAB (the Mesh-Oriented datABase), MSTK (MeSh ToolKit), and SimpleMesh, a useful debugging and testing mesh. The common interface to the four frameworks provides access to local mesh data appropriate for use in the Discretization Toolsets and Process Kernels.

The structured mesh infrastructure uses BoxLib, a block-structured adaptive mesh refinement (AMR) library, to enable the efficient tracking of solution features and accurate modeling of fine-scale spatial features, including engineered structures. On each mesh block the mesh infrastructure presents the data in a structured regular layout that simplifies discretization and metadata operations, such as those related to point-to-point data communication. Thus, while the assumption of block-structured data places restrictions on the geometric generality of the simulation domain, it enables significant algorithm simplification and optimizations within the PK and mesh implementations. It also fundamentally exposes the details of direct data access to the PK and MPC layers.

2.3.2 Discretizations Toolsets

The continuum models of interest here are most often expressed as a system of differential equations that include conservation laws and constitutive relationships involving spatio-temporal differential operators. Additionally, in some discretization methods material properties must be interpolated in space and time in order to construct consistent numerical approximations of the flow and transport models. The Spatial and Temporal Discretization Toolsets provide mesh-specific interfaces to representations of these operators for use in Process Kernels implementations. The toolsets and kernels may be combined to provide a specialized functionality if such constructions are particularly advantageous, as is noted in the discussion of structured discretization methods below. In addition, these toolsets include the Reaction Toolset to simulate the geo-biochemical reactions that are required by the model. In the future it may include additional toolsets such as a Multiscale Toolset to support various multiscale, upscaling and sub-grid algorithms.

Temporal Discretization Toolset

Stiff time-dependent simulations often arise with the Richards Model for unsaturated flow, multiphase flow, and transport. The Temporal Discretization Toolset supports the well-established method-of-lines (MOL) or semi-discretization approach for solving systems of partial differential equations. In MOL the spatial discretization is separated from the time operator, and treated simply as a time-dependent forcing term for a large-scale system of ordinary differential equations (ODEs). The resulting ODEs may be integrated using explicit or implicit methods. Implicit methods are typically better suited for systems exhibiting a wide range of temporal scales, but lead to a large system of coupled nonlinear equations for the discrete system that must be solved with iterative schemes. Explicit schemes are simpler to implement, but are limited typically to the smallest timescales present in the system and may thus be impractical for relatively long-time integrations. The Temporal Discretization Toolset provides access to both implicit and explicit integration approaches, including forward and backward Euler methods and BDF2, and are used for computing time-dependent and steady solutions.

When the time scales of the system are clustered/separated, there are considerable computational advantages to more tightly coupling the spatial and temporal discretizations. For example, when the fluid flow in a system is incompressible, the pressure field becomes elliptically coupled throughout the domain, while fluid advection remains hyperbolic with wave-like propagation that evolves on a finite timescale. A higher-order IMPES-like approach is ideally suited for evolving such flows with minimal discrete phase errors. *Amanzi's* IMPES-like approach is based on an implicit solution of pressure and a time-explicit evolution of the mass conservation equations. It is worth noting that while these IMPES methods can be implemented in the context of nearly any meshing strategy, particularly efficient IMPES methods are possible on structured orthogonal grids, details for which are described in [13].

Spatial Discretization Toolsets

On unstructured meshes Amanzi includes modeling of single-phase Darcy flow in saturated media, and Richards equation in unsaturated media. These models are described by linear and nonlinear elliptic/parabolic PDEs, respectively. To discretize these continuum models a mass conservative (also known as compatible) discretization based on mimetic finite differences is used. This discretization generalizes many popular methods (finite volumes and finite elements) from existing codes and removes many limitations related to grid geometry and parameter anisotropy. In particular, the Mimetic Finite Difference (MFD) method that was designed for convex hexahedral cells [12] has been implemented. Unlike a conventional two-point flux approximation, popular in subsurface modeling due to their simplicity but exhibiting O(1) error on non-orthogonal hexahedral meshes, the MFD method provides a second-order accurate approximation of the pressure. Support for general polyhedral meshes using the appropriate MFD method [5] is being developed.

Purely advective transport of concentration on unstructured meshes is described by a hyperbolic conservation law. On unstructured meshes, a first-order accurate monotone upstream-centered scheme for conservation laws (MUSCL) [19] is provided. This scheme is often referred to as the donor scheme and is commonly used in reactive-transport simulation codes. In addition, a second-order MUSCL scheme with Barth-Jespersen limiters [2] is provided to capture the sharp front of an evolving plume more accurately. This scheme reduces the numerical dispersion significantly and makes the consideration of more general transport models that include hydrodynamic dispersion and molecular diffusion possible. These additional terms are discretized with linear [1, 8] and non-linear [14, 15, 10] MPFA schemes. The motivation for implementing these two types of schemes is to provide flexibility in achieving a discrete maximum principle on general meshes.

For multi-phase flow on structured orthogonal grids, a semi-implicit approach is employed to solve the component densities equations resulting from the IMPES algorithm described earlier. A second-order accurate Godunov method is used to determine the advection term. The Riemann solver used in Amanzi is capable of capturing the characteristic behaviors of the resulting non-linear hyperbolic system accurately [3, 18, 17]. The diffusion term is then discretized based on a Crank-Nicolson scheme. Development and application in the block-structured AMR setting of interest here follows the extension presented in [13]. Similarly, a second-order accurate Godunov method is used to solve the transport equations for the chemical species.

Reaction Toolset

Unlike flow and transport processes, which require numerical representation of spatial gradients and other operations involving multiple grid cells, geochemical and microbiological processes can be described within a discretized control volume (i.e., at a single point). Because of this, it is advantageous to separate the numerical treatment of the reactions from the flow and transport components of the solver. The MPC coordinates the strategy to couple the integration of the geochemical and microbiological processes to other processes.

The Reaction Toolset provides classes to depict chemical species and reactions in a cell in the mesh. The Chemical reactions implemented include equilibrium aqueous complexation, mineral precipitation–dissolution, sorption (i.e. isotherm–based, ion exchange, equilibrium surface complexation), and a general kinetically-controlled reaction for aqueous species to account for sequential decay with daughter products. Also included is a class for activity coefficient calculations. The Beaker class serves as both a container for chemical species and reaction objects and employs Newton's method to solve the resulting reaction ODE. A simple API for the Reaction Toolset was developed to ensure that these processes could be readily coupled to transport and flow models, whether they are represented on structured or unstructured meshes.

2.3.3 Solvers

Linear and nonlinear solvers typically account for a significant fraction of execution time in numerical codes for integrating conservation laws, particularly if the integrations are based on partially or fully implicit temporal discretizations. It is thus critical to streamline these components of the solution as much as possible. In *Amanzi*, this efficiency is achieved by combining a broad range of general purpose solution strategies, with specific solver designs that leverage knowledge of the system and implementations that maximally exploit known structures of the data organization.

The nonlinear solvers in *Amanzi* are based on Newton-type iterative schemes. The unstructured discretizations leverage the nonlinear solver packages provided by Trilinos, which provide implementations of Newton-Krylov (NK) and Jacobian-Free NK (JFNK) algorithms. In addition, a nonlinear Krylov accelerated inexact Newton method was implemented within the the Trilinos nonlinear solver package.

Developing effective approximations of the Jacobian matrix to precondition the Krylov iterations is critical to the efficiency and scalability of these solvers. To this end, knowledge of the coupled processes must be combined with knowledge of the mathematical properties of their discrete representations, and these in turn are intimately coupled to the physical layout of the data. As a result,

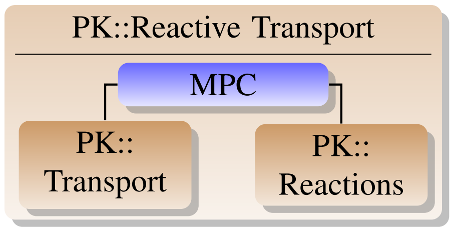

Figure 3: Schematic showing the instantiation of an MPC with weak sequential coupling of transport and reactions as a preconditioner for a fully-implicit Reactive Transport kernel

effective preconditioning strategies may require direct interfaces with the MPC in order to provide a natural and efficient mechanism to logically associate physical processes with matrix sub-blocks of the Jacobian. However, this approach will enable the construction of simplified approximations to the Jacobian through iterative lagging or reduced order discretizations, and generalized linear solver approaches will support the use of different solvers for each block.

Large-sparse linear systems of equations arise through approximations of the Jacobian, or the discretization of a linear processes. For general unstructured meshes, these linear systems are solved with symmetric or nonsymmetric Krylov-based methods, and preconditioned with algebraic multigrid. Using multigrid as a preconditioner is critical to the efficiency and scalability of any system with an elliptic component, such as a pressure equation. Both Krylov solvers and multilevel solvers are available in the Trilinos framework. In the case of linear systems for structured grid representations, geometric multigrid provides a robust and scalable framework for linear solution approaches. These specialized solvers are available in the BoxLib library.

2.4 Core Services

The HPC Core Framework is responsible for the infrastructure that the simulation code will be built on. It is a diverse set of capabilities including build system tools; parallel input and output functionality; uniform error handling and testing; and scripting tools for input file creation. These features link all the parts of *Amanzi*, and define how it interacts with the Platform and Site Applications teams in ASCEM. Consequently, the Amanzi design philosophy strives to create a development environment that is intuitive to use and flexible enough to adapt to the changing needs of end-users and developers over the lifetime of the project. Developers leverage existing tools and software whenever possible with the understanding that providing a solution to meet the project's needs is the priority.

The I/O library is particularly important for *Amanzi*, as it provides output for visualization and input/output for restarts. It creates the restart and visualization files, and sets up their internal structure. It provides a simple interface for the MPC to interact with these files, as the MPC is responsible for managing the restart and visualization files for the simulation. The I/O library supports interfaces to several output file formats, including CGNS (the CFD General Notation System), GMV (General Mesh Viewer), and HDF5 (Hierarchical Data Format version 5). The latter, HDF5 is preferred as parallel reading and writing is only available for this format. Meta data in the Xdmf format, which is uses XML, is written along with the HDF5 file. Visualization packages, such as VisIt, utilize the Xdmf file to navigate the HDF5 file layout. The Xdmf information is managed inside the Output library and automatically written with the HDF5 file.

3 Structured and Unstructured Options

The *Amanzi* design features a dual capability that supports both structured and unstructured mesh options. Although both options target identical capabilities from a mathematical point of view, each approach has distinct numerical advantages. *Amanzi* presents seamless access to both solution approaches, allowing the end-user to select the option that is most appropriate for the requirements of the specific problem at hand. It is important to acknowledge that the assumptions regarding the data layout play a key role in the implementation of the various components. The sections below highlight the advantages, key design elements, and the workflow of each meshing option.

3.1 Structured Grid

The structured grid component of *Amanzi* is based on the BoxLib [4] implementation of Adaptive Mesh Refinement (AMR). BoxLib is an extensive library of C++ and Fortran classes and functions to support AMR calculations on modern distributed memory massively parallel computing architectures. In AMR, a subregion of the computational domain is represented as the union of rectangular grid patches, each of which consists of cells with a uniform grid spacing that is smaller (by a factor of two or four in each dimension) than the cells of the underlying coarse cells. The union of rectangular subgrids which constitute the fine level mesh overlay coarse cells in regions where local errors are deemed too high. Additional refinement levels are added dynamically until the solution is adequately resolved. Each subgrid of a level consists of a logically rectangular block of cells, typically 32-64 per dimension. A large simulation may contain tens of thousands of subgrids. Rectangular subgrids enable the use of high-performance PDE integration methods whose convergence properties are well understood.

In the AMR BoxLib design the nested refinements have boundaries that coincide with the grid lines of the underlying coarse mesh. This feature greatly simplifies the maintenance of key numerical properties of PDE discretizations, such as discrete conservation of numerical flux approximations. On a given AMR refinement level, the boxes exist in a global index space, which greatly simplifies many *metadata* operations, such as the evaluation of box intersections, data communication, and transferring data across refinement levels (i.e. interpolation, averaging). This also greatly simplifies the interface to the raw data at the lowest levels of the implementations, such as the evaluation of gradient discretizations.

AMR is specifically designed for efficient numerical evolution of PDE-based models that can be expressed in conservation form. The hierarchical mesh system is constructed in a telescoping style, where refined coarse cells and their faces are exactly overlayed by fine cells at the next level. In this composite mesh hierarchy, the solution at any point in space is taken from the finest cell covering that location.

BoxLib is loosely built on an object-oriented paradigm, where the details of the data structure and operations on that data are "hidden" or encapsulated into object classes. The BoxLib foundation classes provide a number of optimized data structures for scalable distribution and manipulation of block-structured data. In addition, the data structures include a highly tuned set of routines for portable, scalable I/O, and a generalized interface to dynamic load and memory balancing.

Extensive experience with this design has shown the need for a hybrid approach that exposes some of the underlying formats to simultaneously allow convenient expression and extreme efficiency on modern computing architectures. The BoxLib library is also hybrid in the sense that C++ is used primarily for dynamic memory management and control flow, while the numerically intensive portions of the code are typically expressed in FORTRAN.

The fundamental parallel data container in BoxLib is the MultiFab, which is the class that encapsulates FORTRAN compatible arrays defined on unions of block-structured data chunks. The grids that make up a MultiFab are distributed among the processors using a dynamic distribution strategy which also caches information necessary for efficient message passing of ghost cell data.

To maximize portability of the library, BoxLib currently uses only core MPI functionality. Moreover, the MPI specific library calls are encapsulated within a class that presents an abstraction of the message passing environment. The abstraction facilitates porting of the library to other message passing environments as well.

3.2 Unstructured Grid

In the last decade unstructured mesh toolkits and libraries have progressed significantly, making the development of large-scale parallel simulation capabilities viable. Recently there has been growing support in these mesh toolkits for polygonal and polyhedral elements, which are ideal for capturing complex geometric features common in subsurface environments. The Amanzi mesh infrastructure is designed as a lightweight layer that wraps several mesh toolkits in a common API. The supported mesh toolkits include the Mesh-Oriented datABase (MOAB), the MeSh ToolKit (MSTK), and the Sierra Tool Kit mesh (STKmesh) from Trilinos. In addition, a simple structured hex-mesh generation capability is provided through STKmesh to facilitate simulations with simple geometries.

The key features of the unstructured mesh infrastructure, AmanziMesh, are readily illustrated by stepping through various aspects of defining, discretizing, and solving a flow and transport problem. The natural starting point is for the AmanziMesh to read an unstructured mesh. These unstructured meshes are typically generated by the Platform Model Setup tools, but could come from any tool that can write a mesh using the ExodusII format [9] (e.g., Cubit [7]). This format now supports polygonal and polyhedral elements, and is readily partitioned across a distributed memory machine using conversion utilities included with Trilinos. As the mesh file is read by AmanziMesh on each processor, various data structures containing explicit representations of cells (highest level mesh entities), nodes (lowest level mesh entities) and faces (intermediate level mesh entities bridging two cells) are created. In addition, it builds the local and global indexing of all mesh entities, and establishes the required connectivity information.

To create a PK on this unstructured mesh various sets of faces and cells are required. First, to define the boundary conditions regions of the boundary surface must be specified. These sets of boundary faces are called "side sets", and may be included in the mesh file, or specified in the input file using geometric constraints. Similarly, parameters for the PK to use on specific interior regions must be defined. These sets are called ”cell sets", and they too may be specified in the mesh file, or through regions tag in the input file. To take advantage of various packages in Trilinos, the model parameters and solution are stored in native distributed Trilinos vectors or multi-vectors. AmanziMesh builds the communication maps for these ePetra vectors as it reads the mesh. With these “side sets”, “cell sets”, and ePetra maps defined, the discretization toolsets can create the discrete systems of equations that will be evolved in time for this simulation. In the case of implicit integration or steady-state calculations, the systems are solved by the Trilinos nonlinear solver, NOX.

Spatial discretizations were another fundamental hurdle for unstructured meshes, because grid distortion can destroy important properties of the discrete operators (e.g., accuracy and monotonicity). But significant progress has been made on these discretizations in the last decade. This is particularly true for the mimetic finite difference (MFD) methods highlighted above, which have matured to handle general polygonal/polyhedral cells. In fact, these cells are not required to be convex, making these methods ideal for logically structured grids. In addition, these discretizations can be optimized through free-parameter selection to preserve the discrete maximum principle in many cases.

3.3 Common source

The dual mesh option creates very little overhead from a code development perspective, while its richer development and capability environment is a significant advantage for keeping pace with emerging architectures. Specifically, most large mesh dependent components or libraries, such as the mesh toolkits and multigrid solvers, are available as third party libraries. Also, in a flexible simulation code such as Amanzi, a significant amount of code is devoted to mesh independent tasks. For example, parsing and verifying input files, and interfaces to community standardized specifications of material properties, such as GSLib. In the current implementation of Amanzi, the MPC weakly (sequentially) couples the Transport PK and the Reaction PK. In this case the the reaction network appears in the form of a pure ODE with no spatial dependence, and hence, no explicit dependence on the spatial layout of the data. Hence the identical Reaction PK is used in both the structured and unstructured Amanzi simulations. It is highly beneficial to identify these opportunities that both reduce coding redundancy and ensure a level consistency between the meshing options.

4 Third-Party Libraries

In order to accelerate the software development of ASCEM, one of the main design strategies is to leverage existing capabilities wherever possible. In fact, many capabilities in the form of software frameworks have already been developed by other DOE programs and it makes sense to take advantage of these efforts. Here, a framework describes a software layer that provides basic functionality to a code project in a consistent and clearly defined manner. Almost all modern software designs use some type of framework and ASCEM is no exception. However, our design is also heavily dependent on the use of third-party libraries (TPLs) to build the *Amanzi* framework.

Using a TPL is an appealing way to reduce costs and quickly launch a project. It reinforces the concept of a modular design and allows developers to focus on algorithm development and not spend valuable time on a capability or functionality that has already been implemented. However, using a TPL framework can also be prescriptive. It should be noted that accepting third-party software into a project is not free; in fact, it may just transfer development costs to a future point in the software life-cycle. These costs are incurred as the host code matures and it must continue to comply with the standards and assumptions that the third-party framework imposes on its users. In addition, the host code may have to determine the appropriate course of action should a TPL stop being developed or supported.

An example of this, is the prevalent use of the Message Passing Interface (MPI) standard in parallel HPC codes. This standard has allowed the development of parallel algorithms that are distributed across thousands of compute resources. Without MPI, every code would need a software layer to communicate between processes and this software would require significant time to port each time a code was ported to a new machine. By accepting the MPI standard and designing process communication around the MPI APIs, code projects transferred the time spent developing communication software to developing scientific software. However, the MPI standard assumes that the computing environment is homogeneous and every code that depends on MPI as a communication library also makes this same assumption. With emergence of heterogeneous computing hardware, some of the time and money saved using MPI must now be reinvested to redesign and refactor codes for emerging computer architectures.

This example also demonstrates another hazard of using TPLs in any project. Codes with explicit MPI function calls throughout the source will require a difficult and costly refactoring to support a heterogeneous computing environment. Alternatively, establishing an interface layer that bridges the host code to the library (i.e., a wrapper) serves to isolate the third-party functions. This isolation creates a design that the host code can manage and adapt more efficiently. In *Amanzi*, interface layers are used to minimize direct calls to TPL interfaces and routines.

The incorporation of third-party libraries (TPLs) is an inevitable aspect of efficiently constructing a large software framework. There is a delicate balance between the benefits of incorporating a well-designed TPL, and the need to accept the organization it may impose on the structure of the code that uses them. In some cases, adoption of a TPL will be a temporary approach for prototyping designs/implementations. In other cases, the TPL's will provide key functionality for *Amanzi*. Trilinos, BoxLib and many of the other software packages used in *Amanzi* are examples of the latter.

5 Computer Architectures

The *Amanzi* simulator must run on a diverse set of hardware configurations and operating system environments for it to be useful to the large customer base that it must support. In particular, the project's build system that generates libraries, binaries and test suites must work seamlessly between UNIX-like and Windows systems. Towards this goal, the *Amanzi* project has decided to use the CMake open source build system from Kitware [6], which is committed to platform independence. CMake parses simple text configuration files that generate native build files such as UNIX Makefiles, Eclipse project files, or XCode project files. An additional benefit is the Kitware CTest system, which integrates testing tools directly into any CMake build system. *Amanzi* will

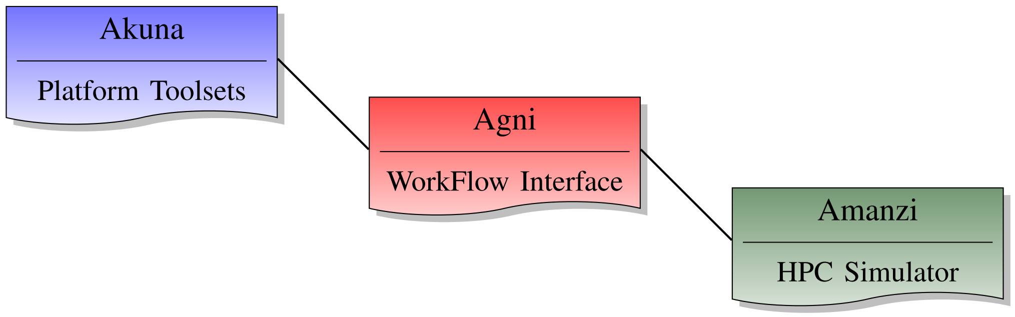

Figure 4: Schematic showing the relationship of the Platform Toolsets, Akuna and Agni, with the HPC Simulator, Amanzi.

use both of these systems to provide a consistent process for adding and removing source files and for creating tests. The last feature is crucial for the verification tasks. In particular, this will encourage developers to write and execute tests as a part of their normal code development work flow and give users confidence that the code is generating consistent and correct answers.

6 Interaction with Platform

ASCEM's Platform Toolset is responsible for packaging the problem-specific parameters discussed above, and then launching the *Amanzi* simulator executable along with these instructions. *Amanzi* in turn evolves the model processes, generates the requested output, and returns control to Platform.

Interaction with the Platform and Integrated Toolsets software is managed by a special component dubbed *Agni*. The current mechanism for communication between the two thrusts is through the exchange of XML (Extensible Markup Language) files containing problem descriptions. This facilitates the recording of simulation parameters by giving it a representation outside of each software component. It also allows us to draw on the extensive collection of libraries and resources for creating, parsing and manipulating XML files.

6.1 Problem setup

The *Agni* interface connecting ASCEM's Platform toolset, *Akuna* to *Amanzi* is sufficiently robust to convey the full set of parameters required to carry out a simulation. This includes boundary condition specifications (functional forms and specific associated values), initialization information, material properties, the chemical model, and specific execution instructions. The chemical model includes the definition of chemical species and their phases. Execution instructions include generic parameters such as simulation time, and mesh-specific instructions, such as AMR regridding frequency.

Geometrical *regions* provide a fundamental abstraction for communicating run-specific instructions from Platform to *Amanzi*. *Regions* are mathematical descriptions of geometrical entities, such as points, lines, planes, surfaces and volumes, and are used to identify the bounding surfaces of the simulation domain and locations for output diagnostics.

Akuna also specifies the set of quantities required for output, including plotting and analysis data (possibly including the derivation of discretization-specific diagnostics), execution statistics, checkpoint/restart instructions, and a special subset of data labeled *observations*, that will be used for uncertainty quantification and sensitivity analyses, and decision support. It should be noted that while control of the output (frequency/volume, quantity selection) is passed via Platform, analysis and checkpoint data in particular are assumed to be "large". The *Amanzi* specification for large data incorporates multiple formats allowing high-performance implementations to be tailored to specific features of the meshing strategy.

Observations are a special class of output data returned through the Plaform-*Amanzi* interface. These observations consist of a small set of basic values (reals, integers, strings) that represent a summary of the simulation that is specific to a particular study and a specific model. It is envisioned that an important execution mode for *Amanzi* will be geared for the construction of surrogate models or other parameter estimation or uncertainty analysis related activities. In this case, only a handful of key diagnostic quantities, rather than the entire computed flow field, will need to be returned to the Platform. Observations involve point-level diagnostics, or integrals of field quantities over particular geometric *regions*, and are transmitted in a format that is generic to the choice of meshing strategy.

Akuna and *Amanzi* are tightly integrated with respect to a common understanding of the various parameters implicit within this specification. In particular, a finite set of generic parameterized geometrical features and boundary condition functional forms are supported. Material properties may be communicated through several options. One example might be to use pointers to files in the community standard GSLib file format. Other options are also being investigated. The format and transmission procedures for observations is not yet established. Large data files for detailed analysis and execution restart are not returned directly to *Akuna*, although their format will be compatible with the ASCEM data analysis toolsets.

7 Conclusions

In this document the high-level design of Amanzi was presented. The discussion focused on the motivation and components of the object-oriented hierarchical design and its alignment with the processes being modeled and simulated. This design ensures that Amanzi is both a flexible and extensible simulator suitable for use and development by this broad community. In particular, the Multi-Process Coordinator (MPC) was introduced as well as the discrete representation of the process models, namely Process Kernels (PKs). The importance and flexibility of the dual structured/unstructured grid approach was discussed, along with the supporting toolsets, namely, the mesh infrastructure, discretizations, reactions, and solvers, were highlighted. Key mesh infrastructure and discretization methods were highlighted, and the motivation and critical role of leveraging Third Party Libraries (TPLs) was presented. However, implementation issues such as languages, interfaces, and data structures are beyond the scope of this high-level design document and will be presented in a upcoming Amanzi documentation. Finally, A high-level description of the interaction of Amanzi with the Platform Toolset Akuna was discussed.

REFERENCES

[1] I. Aavatsmark. An introduction to multipoint flux approximations for quadrilateral grids. *Comp. Geosciences*, 6:405–432, 2002.

[2] T. J. Barth and D. D. Jespersen. The design and application of upwind schemes on unstructured meshes. *AIAA*, paper 89-0366, Jan 1989.

[3] J. B. Bell, P. Colella, and J. A. Trangenstein. Higher-order Godunov methods for general systems of hyperbolic conservation laws. *JCP*, 82(2):362–397, June 1989.

[4] Boxlib: A block-structured amr framework. https://ccse.lbl.gov/BoxLib/index.html.

[5] F. Brezzi, K. Lipnikov, M Shashkov, and V. Simoncini. A new discretization methodology for diffusion problems on generalized polyhedral meshes. *Comput. Methods Appl. Mech. Engrg.*, 196:3682–3692, 2007.

[6] Cmake, the cross-platform build system. http://cmake.org.

[7] Cubit: Geometry and mesh generation toolkit. http://cubit.sandia.gov/.

[8] Michael G. Edwards and Hongwen Zheng. Double-families of quasi-positive darcy-flux approximations with highly anisotropic tensors on structured and unstructured grids. *J. Comput. Phys.*, 229(3):594–625, 2010.

[9] Exodus ii: A finite element data model. http://sourceforge.net/projects/exodusii.

[10] K Lipnikov, D Svyatskiy, and Y Vassilevski. A monotone finite volume method for advection-diffusion equations on unstructured polygonal meshes. *J. Comput. Phys.*, 229(11):4017–32, 2010.

[11] Robert C. Martin. *Agile Software Development: Principles, Patterns, and Practices*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2003.

[12] J. Morel, R. Roberts, and M. Shashkov. A local support-operators diffusion discretization scheme for quadrilateral $r-z$ meshes. *J. Comput. Phys.*, 144:17–51, 1998.

[13] G. S. H. Pau, A. S. Almgren, J. B. Bell, and M. J. Lijewski. A parallel second-order adaptive mesh algorithm for incompressible flow in porous media. *Phil. Trans. R. Soc. A*, 367:4633–4654, 2009.

[14] C. Le Potier. Finite volume scheme satisfying maximum and minimum principles for anisotropic diffusion operators. In R. Eymard and J.-M. Herard, editors, *Finite Volumes for Complex Applications V*, pages 103–118, 2008.

[15] Zhiqiang Sheng and Guangwei Yuan. The finite volume scheme preserving extremum principle for diffusion equations on polygonal meshes. *J. Comput. Phys.*, 230(7):2588–2604, 2011.

[16] C. Steefel, D. Moulton, and P. Lichtner et al. Mathematical formulation requirements and specifications for the process models. Technical Report ASCEM-HPC-2011-01-0a, Lawerence Berkeley National Laboratory, 2011.

[17] J. A. Trangenstein and J. B. Bell. Mathematical structure of black-oil reservoir simulation. *SIAM J. Appl. Math.*, 49:749–783, 1989.

[18] J. A. Trangenstein and J. B. Bell. Mathematical structure of compositional reservoir simulation. *SIAM J. Sci. Stat. Comput.*, 10:817–845, 1989.

[19] B. van Leer. Towards the ultimate conservative difference scheme, v. a second order sequel to godunov's method. *J. Comput. Phys.*, 32(1):101–136, 1979.